commodore

PET USERS CLUB NEWSLETTER

Volume 2 Issue No. 2

Subscriptions:-

Commodore Systems Division,
Users Club Subscriptions Department,
818 Leigh Road,
SLOUGH,
Berkshire.

Editorial:-

The Editor CPUCN,
Commodore Information Centre,
360, Euston Road,
London, NW1.

EDITORIAL

After editing six out of the last eight issues, Richard Pawson has left the "Pet Users Club Newsletter". We wish him well with his new venture, "Printout", and with his career at Warwick University. During the coming year we will build on the foundation established by Richard. We will continue to increase the space given to articles and programs, and the ratio of advertising to "hard copy" will be carefully controlled.

Although we refer to annual subscriptions, most of you will be aware that your first year in fact covered 18 months of issues in volume one to ensure you value for money. This is the second edition of volume two and we intend to give you 8 issues before your next renewal.

This autumn we were again visited by that transatlantic PET wizard, Jim Butterfield. Jim gave a most fascinating and entertaining lecture to the North London Hobby Computer Club - and much of the material he brought over with him is reproduced in this newsletter. Unfortunately, we cannot rely on Jim's occasional forays into the U.K. to provide material for every edition of the newsletter - so come on everybody let's show the world that the U.K. has its PET men too! In order to stimulate prospective authors to reach for their pens, we will be providing a £50 voucher, exchangeable for Commodore Products to the author of the best article or program published each month. In addition all articles and programs published in Volume II of the newsletter will be eligible for the "Best of the Year" competition with a £250 Commodore voucher as its prize.

We will, of course, always continue to give you up-to-the-minute news and reviews of Commodore's latest hardware and software developments, and articles from Commodore's own personnel. Newsletter publishing apart, what should the future role of the U.K. Pet Users Club be? Should we form a number of regional groups which would meet periodically and host guest speakers from the PET world? Or, should we concentrate on publishing the neswletter and be content with providing a forum for independent groups such as the North London Hobby Computer Club and the Independent Pet Users Club to advertise their own social activities.

Write to us with your thoughts - future development of the Pet Users Club will depend on your ideas and willingness to help transform such ideas into action.

ANDREW GOLTZ

COMMODORE NEWS

Commodore's official Business Software, designed to run on our own floppy disc units, was publically unveiled at the International Business Show in late October. CSTOCK, CBIS, WORD PROCESSOR 1, and the new PAYROLL are now available from selected "Business Software" dealers. The programs are designed for the businessman who requires powerful ready-to-run packages. Program development has been very thorough - during three weeks continuous demonstration at I.B.S and Compec no programming bugs appeared. These programs demonstrate Commodore's continuing committment to produce high quality products and to give the customer the best possible value for money. CSTOCK and CBIS are £150 each. WORD PROCESSOR 1 (in the opinion of those who have seen it already is a generation ahead of anything so far seen on PET) is only £75 and the new improved Payroll (with coin analysis)is on cassette at £50. All prices are exclusive of VAT. Commodore Business Software is being supported by Commodore Business Software dealers who have been specially selected from our national official dealer network on their ability to provide a total service to the business community. They are equipped to handle analysis, software selection, software modification and installation as well as provide operator training where required. Please note these additional services are not included in the standard package price.

Simultaneously to the launch of a new generation of Business Software to run on the Commodore Disk Unit, we have released two important programs for the serious programmer. The Official Commodore Assembler provides development capabilities never before available for the PET and is reviewed in depth elsewhere in this issue. LISP, an important high-level language used in the field of Artificial Intelligence, considerably extends PET's utility as a research tool. Both programs require the Commodore Floppy Disk unit to operate.

To meet the extremely heavy demand for the new 32K PETs, Floppy Disk Units, and printers, we are investing in further production capability. The new PETs, have had a superb reception from new owners and former 8K owners alike, and are available almost "off the shelf" from dealers. There is a queue for Floppy Disk Units and Printers although we are rapidly reducing this and we are currently quoting dealers about 45 days delivery on new orders. We expect to have cut this down to 30 days by the new year.

NEW 1980 TRAINING COURSE PROGRAM

Since announcing the start of our Commodore training courses we have received many enquiries regarding 1980 courses, dates and plans. We have also had many delighted participants go through the first courses. Where possible we have tried to take account of your requests and have extended the scope of courses offered. In addition to our intensive residential training courses we are including some one day seminars on popular topics. The courses and seminars can be booked through your local dealer or on a payment with order basis direct to:

Commodore Systems,
Training Course Bookings,
818 Leigh Road,
Slough,
Berks.

Bookings will be allocated on a first come basis. Please include an alternative choice with your booking where possible. The seminars include lunch and the training courses full residential costs at first class Trust House Forte Hotels.

TRAINING COURSES FOR 1980

All Commodore training courses provide practical skills. To achieve this they are intensive residential courses with one Pet computer and disk unit for every two students. Practical work forms a significant portion of each course.

The tutors are professional computing instructors. Comprehensive documentation is supplied. The atmosphere is friendly and informal. A great deal of personal tuition is provided during the exercise sessions.

Basic for Beginners

* Aimed at those who want to learn how to program and gain a thorough grounding in BASIC.
* Takes you from scratch to the point where you can write useful commercial and scientific programs.
* Tackles all BASIC commands and statements up to but not including file handling.

This course provides the quickest way we know to learn BASIC. You will end up exhausted but probably very pleased with yourself.

Commodore Disk Programming

* Aimed at those who wish to design systems and BASIC programs which use the Commodore floppy disk system to its full extent.
* Teaches you about files and how to use them in your programs.
* Explains and gives you practical experience in using the wide range of Commodore disk facilities including random access.

Assembly Language Course

* Aimed at all who wish to program the PET or any other 6502 based micro computer in machine code.
* Teaches you the elements of micro processors and how to program them.
* Shows you how to use the power of Assembly Language.

Advanced Basic Course

* Aimed at those who have a good grasp of the commands and statements of BASIC who want to improve and speed up their programming techniques.
* Teaches you a professional approach to programming.
* Deals with data files on cassette.
* Introduces some sophisticated techniques for using such commands as PEEK and POKE.

APPRECIATION SEMINARS FOR 1980

Commodore appreciation seminars will be run by the training department to the same high standard as the training courses. The lectures are given by senior Commodore Managers and guest speakers who are acknowledged experts in their field. Commodore hardware and software packages are used to illustrate the lectures, provide demonstrations and, where appropriate, hands on experience.

Micro Computers and the Business Man

* What can a micro computer do for my business?
* What are the limitations, the costs and timescales?
* What are the advantages, the savings, the ways in which efficiency is improved?

The lectures are supported by demonstrations and hands-on experience of Accounting, Business Information, Payroll, Word-Processing and other business packages.

Micro Computers In Control Systems

* The use of micro computers as programmable monitoring and control tools.
* Interfacing micro computers to industrial and laboratory equipment.
* The micro computer as a development tool for dedicated system software.

COURSE AND SEMINAR SCHEDULE JAN-MAY 1980

BASIC FOR BEGINNERS — Monday to Wednesday (3 days)
£250 + vat

Jan	7- 9	Skyway, Heathrow
Feb	4- 6	Skyway, Heathrow
Mar	3- 5	Skyway, Heathrow
Mar	31-	
Apr	2	Skyway, Heathrow

ADVANCED BASIC PROGRAMMING — Wednesday and Thursday (2 days)
£150 + vat

Feb	13-14	Skyway, Heathrow
Feb	27-28	Excelsior, Birmingham
Mar	12-13	Skyway, Heathrow

COMMODORE DISK PROGRAMMING — Thursday and Friday (1½ days)
£125 + vat

Jan	10-11	Skyway, Heathrow
Feb	7- 8	Skyway, Heathrow
Mar	6- 7	Skyway, Heathrow
May	1- 2	Excelsior, Birmingham

ASSEMBLY LANGUAGE COURSE — Monday to Wednesday (3 days)
£250 + vat

Apr	28-30	Excelsior, Birmingham

MICRO COMPUTERS AND THE BUSINESS MAN — Monday
£50 + vat

Jan 14	Skyway, Heathrow
Feb 11	Skyway, Heathrow
Feb 25	Excelsior, Birmingham
Mar 10	Skyway, Heathrow
Apr 4	Excelsior, Manchester

MICRO COMPUTERS IN CONTROL SYSTEMS — Tuesday
£50 + vat

Jan 15	Skyway, Heathrow
Feb 12	Skyway, Heathrow
Feb 26	Excelsior, Birmingham
Mar 11	Skyway, Heathrow
Apr 15	Excelsior, Manchester

SOFTWARE NOTES

ERRATUM

The fix for garbage collection is proving a success - but the reporting of it in the last issue of CPUCN was not. Firstly, the fix was in line 90 (not 55 as stated) and line 90 should have read:

```
90  POKE 53,N
```

The typo goblins couldn't have struck in a worse place!

NEW PRODUCTS

We have now released our Stock control Package, (CSTOCK), our Business Information System (CBIS) and our Word Processor (WordPro) all of which are available from Business software Dealers.

The assembly Language Development System and LISP have also been released and are available through all dealers. There are technical descriptions of these programs in this magazine and in the Computer Press. We think this Software represents excellent value for money and confirms our lead in producing more for less money. Your dealer may recommend professional installation of Business packages. We would, at this stage, suggest that only the highly competent purchase Business packages without installation by your local dealer. Personal installation is not included in the package cost and your dealer can quote you on this aspect.

RANDOM ACCESS PROGRAMMING

I am still getting requests for more about disk programming. So here are some notes which may be of interest to you, in addition you may find it useful to refer to a standard work: 'Computer Date-Base Organisation' by James Martin, Prentice-Hall, 1975. Martin's works, although IBM orientated, are always extremely readable. Closer to home, we have the program Random 1.0 listed in the back of the disk manual and distributed to you on the demonstration disk. It may seem rather long, but it shows many of the main features of Random Access Programming. In addition there are our own intensive training courses for those wanting to get rapid "driving lessons" in computing techniques.

Using an internal relative record number, it constructs a descriptor file. This is a sequential file of keys obtained from the first field (sub-record delimited by a carriage return). Note that a key is a distinguished portion of text held in a record used to compute, possibly via a "hashing" algorithm, the physical disk address. This file can then be examined for a particular attribute which then points to the relative record concerned. The relative record number is then converted to track and sector addresses on disk.

Random Access Programming of a disk is always a question of producing by whatever means possible, a physical disk address (track, sector and buffer positions in the case of 'packed' records) in response to a request for data.

Often sequential files are kept to either point directly to a record (or even another index) or to contain information about a record (live or deleted) as in this last case for CBIS, CSTOCK, etc, and as does the BAM of system level in the 2/3040 disk.

The most widely used technique is the Index Sequential Method - Martin here is superb. The other principal method of file structuring is linked lists: here sectors contain pointers to the next sectors in the file. It is up to the programmer to decide what technique (or mixtures of techniques) is most desirable for a given application.

To come down from this to basics, if you simply want to get going with BLOCK-READ and BLOCK-WRITE, use the program below. Bear in mind that in any real world application, you will probably be using all sorts of exotic techniques to compute Track and Sector.

```
 10 OPEN1,8,5,"#"
 20 OPEN15,8,15
 30 FORT=1TO3
 40 FORS=0TO20
 50 PRINT#15,"B-P";5;1
 80 X$=STR$((T-1)*21+S)+"TESTING"
 90 PRINTX$
 100 PRINT#1,X$;CHR$(13);
 110 PRINT#15,"B-W";5;1;T;S
 120 NEXT
 130 NEXT
 140 CLOSE1
 150 OPEN2,8,5,"#"
 160 FORT=1TO3
 170 FORS=0TO20
 180 PRINT#15,"B-R";5;1;T;S
 190 INPUT#2,A$:PRINTA$
 200 NEXT
 210 NEXT
 220 CLOSE2:CLOSE15
 230 END
READY.
```

During implementation sub-routine 1000 should be used freely. For business applications error checking should be done on all occasions i.e. after the OPEN's in 1Ø - 2Ø, after the B-W and B-R commands and after the CLOSE 2. Apart from anything else, this will ensure media are behaving reliably. If EN<>Ø then helpful error messages should be printed to the user.

STARTREK COMPETITION

Firstly, I'd like to thank everyone who entered the competition and congratulate you on producing some superbly entertaining programs.

The judging of the entries was based on a number of points - real-time programs were preferred to more static versions, good use of graphics also scored highly and the sheer entertainment value of the program was obviously very important.

The winner that emerged from all this was a program written by Deri James, so congratulations to him (and, incidently, to his wife who has just produced him a new baby).

This program is now available for £10.00 on order No. MPØ51, packaged along with another game called Petopoly, a compulsive variation of the well known Monopoly. Both games are superb, both being written by very good programmers and for £10.00 we think that they provide excellent value for money.

When our new Master Library catalogue went to press, we only had an old ROM version of Startrek, but this has now been changed to run on all machines, so there needn't be any worries about machine compatibility.

In fact, all of Commodore's Master Library, with the exception of just one of the games, Super 9x 9, is machine independant and we hear that even this will be corrected by the time you get your next newsletter. Yet another sign of Commodore going boldly where no manufacturer has gone before!

All our new releases were carefully vetted for this compatability before being announced, and since most of them were in daily use by their authors prior to Commodore releasing them nationwide, I'm sure what university lecturers, programmers, teachers, financial advisors, hobbyists, etc found extremely useful, you will too.

The demand we've had already for these new programs has more than justified their release. Don't get left behind!

Finally, if there's any Software you've got that you think we'd be interested in, please send it to:-

Peter Gerrard,
Cassette Library Manager,
Commodore Business Machines (UK) Ltd.,
818 Leigh Road Trading Estate,
Slough,
Berkshire.

Let's keep Commodore where it belongs - at the top!

THE COMMODORE ASSEMBLER DEVELOPMENT SYSTEM

As many of you will now be aware, there is now available in the U.K. the standard Commodore Assembler Development System. Unfortunately, we have had to release it, at least initially, in a form which only works with 16K and 32K disk-based systems. This is largely because of the powerful .LIB and .FILE pseudo-operators in our assembler, which allow the linking or insertion of multiple source files from disk, an approach abviously not sensible on a cassette based system. Certainly our whole design concept for this assembler was that it should be extremely fast, and capable of assembling very large source files. The Assembler program itself occupies 8K of RAM, leaving us either 8K or 24K of symbol table space, enough for 1000 or 3000 symbols respectively. This system has been used to assemble a source file which generated 16K of object code. As for speed, I timed it for both passes of a fully documented 777 line source file, complete with listing on the screen, at less than two minutes. Certainly with this sort of performance, I feel that although it would be handy on a cassette system, this Assembler is much more appropriately used on a disk-based system.

Let me now just go over some of the major features of our development system. First off, it costs £50, and comes with its diskette and 35-page manual packed in a white multi-ring binder. The major programs you get are an Editor, Assembler, Loader, Hi-Loader, and a superb monitor-debugger. Let's take them in that order!

The editor is a screen-based editor, which seems like (in fact is) the normal PET editor, except for the addition of 11 new commands. These commands include GET and PUT, for loading and saving assembler source files, NUMBER for renumbering, REPEAT for enabling a software repeat key feature, FIND, CHANGE, and AUTO for automatic line numbering on input. Block delete, break to monitor, formatted listing, and editor disable round out the compliment of new functions.

It is of course very pleasant to be able to edit your assembler source programs from within essentially the same friendly environment used to edit BASIC programs. Some of the less obvious advantages include the automatic line numbering feature, and the formatting features built into both the editor and the assembler. As you type source lines into the editor, labelled lines start on column one, unlabelled lines need only a one or more space prefix. One or more spaces are used to separate the various fields of each line of source code, but when it is listed using format, or when the assembler produces a source listing, a uniform formatting is performed which prints out your program beautifully. This feature, combined with the paginated and titled listings, alphabetised symbol table, and various listing-control pseudo-operations, gives very much a big-machine feel and performance to the development system.

The Assembler program itself, and the two Loaders provided, are very much 'black box' programs. They do well-defined tasks quietly and efficiently. The Assembler reads source files from disk, and generates TIM/KIM/MDT format object files on disk. These files may be names as you like, although I would suggest the adoption of a regimen of using - SRC and -OBJ suffixes. Object code may be generated or not, as you like. The source listing may go to the screen or any IEEE printer, with any errors (and .LIB and .FILE operations) always echoed to the screen - barring the use of the NOE (No Errors) assembly-time option.

The Assembler can identify 18 different types of error, and the accompanying manual provides an in-depth description of each. The Assembler also flags the precise location of any error on your source line.

The loaders are used to convert your Assembler program from object code on disk to a binary file in RAM. Because a loader must be resident in RAM itself, two versions are provided which reside at different locations, enabling the user to origin his code at any location he chooses.

Overall the Development System really is a pleasure to use. The speed and reliability with which an assembler program can be updated, re-assembled, and re-loaded is a quantum leap ahead of anything previously available on the PET, and the powerful set of assembly time options and features make it a better development system than is available on most micros currently. But really, the icing on the cake is EXTRAMON 7.5 ⟨PA⟩ . This is an enhanced monitor/debugger which makes development work on the PET very pleasant. It contains a mini-assembler, a disassembler, and the abilities to do true code relocation, single-step execution, and break pointed execution. This combined with the various register-display memory-display, memory-search and memory fill functions means that the PET can now offer a stand-alone development environment which will satisfy even the most exacting user. Given that we have managed to keep the price down to £50, I would expect to see this Assembler selling in an almost one-to-one ratio with our disk drives - in that almost everyone with a disk will want one.

For all of you who have already bought the Assembler, and those of you who do so in the future, good luck and happy programming!

MICHAEL R. WHITEHEAD
Software Product Manager

PS For those of you not familiar with assembler programming I note that our training department is running Assembly Language Courses in 1980.

CBIS - A REVIEW

Ever operated a mailing list and wished to mail everybody on that list who had a special status? Or maintained a property register and needed to obtain immediate details of all properties available in survey with a double garage? Or kept personnel records and required a list of everybody who had worked for the company for longer than five years?

Now all these jobs and hundreds of applications like them can be handled quickly and efficiently using the Commodore Business Information System - CBIS for short.

Essentially a computerized card index, CBIS allows the user to define upto 10 fields per record.

These could be:-

NAME
ADDRESS (Road)
ADDRESS (Town)
ADDRESS (County & Postcode)
TELEPHONE
- - -
- -

or

STOCK NO.
PRODUCT NAME
SUPPLIER NAME
ADDRESS ONE
ADDRESS TWO
ADDRESS THREE
CONTACT
TELEPHONE
LAST PRICE QUOTED
- - -

or any other set of field names you want.

Once the user has defined his fields upto 650 records can be stored per disk, and instantly recalled. Upto 25 characters can be recorded in each field, giving a total of 250 characters storage per record.

When the database has been entered the program really begins to get clever. As well as displaying all the records or printing out a complete list on the Commodore Printer, selective searches can be made.

"a list of all sales prospects in Huddersfield" ... or "the names and addresses of all suppliers of ball bearings" - can be immediately obtained by keying the appropriate parameter. Upto 10 parameters can be specified at one time e.g. "properties in SURREY, with FOUR BEDROOMS, TWO GARAGES, near a RAILWAY STATION" . . . and so on. Numeric comparisons can also be easily made:- "more than TWO BEDROOMS", "under £100,000".

CBIS is menu-driven, and practically self documenting. Typically a secretary or clerk will require about one day's practice to become completely familiar with the program. Like all Commodore disk software the program is accompanied by an extremely comprehensive operating manual.

CBIS is already being used inside Commodore for making out our list of P.U.C. members and printing the labels for newsletter envelopes - it will undoubtedly become one of the most popular programs in the Business Software Range.

PROGRAMMING

"Where'd the penny go?"

Jim Butterfield
Toronto

PET is certainly the greatest business tool since electric pencil sharpeners, and printers and floppy disks will herald an explosion of commercial applications.

Basic seems like the ideal language for a small business system - but it has a hidden "gotcha" that will give you problems if you don't know how to handle it. I call it, "the missing pennies problem", and it's common to almost all Basic implementations.

Crank up your PET and try this: PRINT 2.23 - 2.18 - - it's a simple business calculation and the answer has gotta be a nickel, right? So how come PET says .0499999998?

Think of the mess this could cause if you're printing out neat columns of pounds and pence results. Think of the problems if you arrange to print the first two places behind the decimal point: you'll print .04 instead of .05! Think of what the auditor will say when he finds that the totals don't add up correctly!

In a moment we'll discuss how to get rid of this problem. First, though, let's see how it happens.

PET holds numbers in floating binary. That means certain fractions don't work out evenly. Just as, in decimal, one third works out to .333333..., an endless number, PET sees fractions like .10 or .68 as endless repeating fractions - in binary. To fit the fraction in memory, it must trim it. Thus, many fractions such as .37 are adjusted slightly before storage.

Try this program: it will tell you how numbers are stored inside PET:

```
1ØØ  INPUT"AMOUNT";A:B=INT(A):C=A-B:?A;"=";B;" ";
11Ø  FORJ=1TO1Ø:C=C*1Ø:D=INT(C):C=C-D:?D;:IFC> Ø THEN NEXTJ
12Ø  ?:GOTO1ØØ
```

If you try entering numbers in our above example, 2.23 and 2.18, you'll see how PET stores them - and why the problems happen.

How to fix the problem. Easy. Change all numbers to pennies - which eliminates fractions - and your troubles disappear. For example:

```
34Ø  INPUT"AMOUNT";A : A=INT (A*1ØØ + .5) converts A to pennies;

76Ø  PRINT A/1ØØ      outputs pennies in pounds and pence.
```

A Fast Sort

Jim Butterfield
Toronto

When you need to sort a large array, sorting speed becomes important. Most simple sorts become very slow, since twice as many items will take four times as long to sort.

This fast sort is called "selective replacement"; it's classified as a tree type sort. It needs an index array, called I(J) here, which is twice the size of the items to be sorted. Memory can be saved, if needed, by making it an integer type array.

```
100   DIM I(200), N$(100)
110   REM SIMPLE INPUT ROUTINE - WRITE YOUR OWN FOR FILES
120   INPUT "HOW MANY ITEMS"; N
130      FOR J=0 TO N-1
140      INPUT "NAME";N$(J)
150      INPUT "ADDRESS";A$(J)
160      REM  INPUT OTHER DATA HERE IF DESIRED
170      NEXT J
200   REM  SORT STARTS HERE - INITIAL SCAN FINDS FIRST NUMBER
210   B=N-1 : FOR J=0 TO B : I(J)=J : NEXT J
220      FOR J=0 TO N*2-3 STEP 2
230      B=B+1 : I1=I(J) : I2=I(J+1)
240      GOSUB 700  :  REM  PERFORM COMPARISON
250      I(B)=I : NEXT J
300   REM  MAIN LOOP - OUTPUT NEXT VALUE
310   X=X-1 : C=I(B) : IF C <0 GOTO 800
320   REM  OUTPUT ITEM TO SCREEN, PRINTER, OR FILE AS DESIRED
330   PRINT N$(C),A$(C)
340   I(C)=X
350   REM  INNER LOOP TO FIND NEXT ITEM
360   C%=C/2 : J=C%*2 : C=N+C% : IF C > B GOTO 300
370   I1=I(J) : I2=I(J+1)
380   If I1 < 0 THEN I=I2 : GOTO 410
390   IF I2 < 0 THEN I=I1 : GOTO 410
400   GOSUB 700 : REM   PERFORM COMPARISON
410   I(C)=I : GOTO 350
700   REM   COMPARE TWO ITEMS - MODIFY TO FIT APPLICATION
710   I=I1 : IF N$(I2) < N$(I1) THEN I=I2
720   RETURN
800   STOP : REM END OF SORT
```

As you get the sorted item at line 320, it's best to output it (or process it) on the spot. If some reason exists for completing the sort before going on to other processing, you'll find that index array I(J) contains information about the proper order for the data.

COMPUTED GOTO

Ever wanted to program a GOTO followed by an expression such as

```
120 IF ST GOTO (ST * 10)
```

Normally PET does not allow this but Brad Templeton of Mississauss has submitted a machine language routine that will handle a computed GOTO . The program fits in only 12 bytes and can be placed in any part of memory where it won't get clobbered by BASIC. It accesses code in ROM and therefore has two versions, one for original ROM and another for upgrade ROM.

```
Original ROM    JSR CE11      checks for comma
                              else SYNTAX ERROR
                JSR CCA4      evaluates expression
                JSR D6D0      integer? >=0 and <=63999
                JMP C7A0      jump to GOTO routine with result

Upgrade ROM     JSR CDF8
                JSR CC8B      same as above
                JSR D6D2
                JMP C7B0
```

Because the program has no reference to itself, it can be placed anywhere, but for now we'll put it in the second cassette buffer at 826 or hex 033A. Syntax for using the routine will be

```
                SYS826,expression
or...  G%=826   SYSG%,expression

e.g.   IF ST THEN SYS G%, ST* 10
```

BASIC Loader

With the following modification, both of the above routines can be loaded into the second cassette buffer and PET will decide which to use. This way programs using the computed GOTO can be run with either ROMS.

```
                LDA $F202
                BMI $0D
N.ROMS          JSR CDF8
                JSR CC8B
                JSR D6D2
                JMP C7B0
O.ROMS          JSR CE11
                JSR CCA4
                JSR D6D0
                JMP 07A0
```

The following BASIC program will load the above;

```
100  FOR J = 826 TO 854
110  READ X
120  POKE J, X
130  NEXT
200  DATA 173, 02, 242, 48, 13
210  DATA 32, 246, 205, 32, 139, 204,
     32, 210, 214, 76, 176, 199
220  DATA 32, 17, 206, 32, 164, 204,
     32, 208, 214, 76, 160, 199
```

Test with the following;

```
10 G% = 826
20 ?"TEST"
30 SYS G%, 2 * 10
```

Disabling the STOP Key

It's useful to be able to disable the STOP key, so that a program cannot be accidentally (or deliberately) stopped.

METHOD A is quick. Any cassette tape activity will reset the STOP key to its normal function however.

METHOD A, Original ROM:

Disable the STOP key with POKE 537,136
Restore the STOP key with POKE 537,133

METHOD A, Upgrade ROM:

Disable the STOP key with POKE 144,49
Restore the STOP key with POKE 144,46

Method A also disconnects the computer's clocks (TI and TI$). If you need these for timing in your program, you should use method B.

METHOD B, Original ROM:

```
100 R$="20 > :??:9??8=09024 < 88 > 6"
110 FOR I=1 TO LEN(R$)/2
120 POKE I+900,ASC(MID$(R$,I*2-1))*16+
        ASC(MID$(R$,I*2))-816 : NEXT I
```

After the above has run:
Disable the STOP key with POKE 538,3
Restore the STOP key with POKE 538,230

METHOD B, Upgrade ROM:

```
100 R$="20 > :??:9??8=9;004 < 31 > 6"
110 FOR I=1 TO LEN(R$)/2
120 POKE I+844,ASC(MID$(R$,I*2-1))*16+
        ASC(MID$(R$,I*2))-816 : NEXT I
```

After the above has run:
Disable the STOP key with POKE 144,77 : POKE 145,3
Restore the STOP key with POKE 145,230 : POKE 144,46

How they work: Both methods change the interuppt program which takes care of the keyboard, cursor, clocks and the stop key.

Method A simply skips the clock update and the stop key test.

Method B builds a small program into the second cassette buffer which performs the clock update and stop key test, but then nullifies the result of this test.

The little program in method B is contained in R$ in "pig hexadecimal" format. Machine language programmers would read this as:

```
20 EA FF (do clock update and stop key test)
A9 FF 8D 9B 00 (cancel stop test result)
4C 31 E6 (continue with keyboard service, etc.)
```

BEGINNING MACHINE CODE - Fundamental Concepts

This issue we look at fundamental concepts - different kinds of memory; addressing; and jumping to PET's built in machine code routines using the SYS command. Addresses are given for both old ROM and new ROM PET's, the latter in brackets.

ADDRESSING

Every memory location in your PET contains one byte of information. In order for PET to get at these bytes it must have a means of accessing them. Therefore each and every memory location has its own individual address; all 65536 of them. The microprocessor places these addresses on the address buss which immediately enables one memory location to the data buss. Bearing that in mind, one of two operations can happen now. PET can either place a byte into that location (i.e. POKE) or "look" at what's already there (i.e. PEEK). When performing the first operation the microprocessor places a byte on the data buss and transfers it along the buss and into the enabled memory location.

In the second operation, the information or byte in the enabled location is transferred onto the data buss and along the data buss back to the microprocessor. This location is not "emptied" but rather only a duplicate or copy of the information is transferred. Once either of these operations is complete the microprocessor then places a new address on the address buss and another location is enabled. This process repeats thousands of times every second, however these operations aren't possible on all memory locations, but I will explain this later.

The microprocessor has control of 99.9% of the addresses being placed on the address buss. That extra 0.1% control was left for the user and can be obtained through use of the PEEK, POKE and SYS commands. When executing these commands the user must choose an address. This address will be one of the 65,536 memory locations (i.e. 0 to 65535). This is where the memory map enters the picture. The memory map may well be your most powerful tool for choosing addresses. If you look at the map you will see that all of the addresses are listed in ascending order down the left hand side; first in hexadecimal and then in decimal.

(See section on hexadecimal and binary for explanation of this conversion) the decimal address is the one you use when executing the above 3 BASIC commands. To the right are the descriptions of what you can expect to find at the corresponding addresses. If we then PEEK these addresses we are returned the actual bytes that are in those particular memory locations. For example, let us say during a program we hit the STOP key and got:

```
BREAK IN 600
READY.
```

PET gets '600' from a storage register at addresses 138 and 139 (54 and 55). We could also PEEK these locations and find that 600 is indeed stored in 138, 139 (54, 55). However it is not stored as a six, a zero and a zero. Instead it is stored as the decimal conversion of the line numbers representation in hexadecimal. All information of this type is returned in this manner. Now that we know what the memory map will help us do let us cover some of the rules.

RAM and ROM

We all go through life with basically 3 types of memory:

1. MEMORY PRESENT: This memory we use to remember things like what street we're driving on or our present location.

2. MEMORY PERMANENT: Things like our names and fire is hot we never forget.

3. MEMORY PAST: Recent occurrences and not so recent such as things we did 10 or 12 years ago.

In the PET there are only two:

1. RAM Random Access Memory: this type of storage is used for our programs and things that change such as the clock updating routines and loading routines to name a few. These functions would have to be programmed into PET on each power up if they weren't permanently 'burnt in'.

The third type, memory past, is instantly 'forgotten' on power down. The only way to recall it is to first save it on tape, disc, etc.

Recall earlier I mentioned that POKE and PEEK are not possible on all memory locations for several reasons:

A. Not all PET memory locations actually exist. On the memory map, locations 1024 to 32767 is the 'available RAM including expansion'. If you have a PET with 8K, simple arithmetic shows tha ¾ of the available RAM space is non-existent. If you decide to expand your system, PET will 'fit' the added RAM into this area. However, POKing or PEEKing this space (i.e. 8192 to 32767) will return invalid results on 8K PETs

B. The same concept applies to locations 36864 to 49151. This is the available ROM expansion area.

C. Next on the memory map is the Microsoft BASIC area; locations 49152 to 57463. This is the memory that recognizes and performs your commands. Changing the contents of these locations is impossible because it is Read Only Memory and is actually 'burnt in' at the factory. Therefore, POKing these locations will simply do nothing. Also, Microsoft requested that these locations return zeros if PEEKed (for copyright reasons).

With these three rules and your memory map you are now equipped to explore capabilities of your PET that you probably never thought possible. Before we try some examples let's go into one more important occurrence that may have had you scratching your head since that first power up.

MISSING MEMORY?

When you turn on your 8K (where K = 1024) PET, the first thing it tells you is 7167 BYTES FREE; a reduction of almost 12%, similarly a 32K PET displays only 31743 BYTES FREE.

Q. Where did the missing 1024 bytes go?

A. It's still there...right below the available RAM space (notice it starts at location 1024). PET uses this memory to do some very useful operations for you which you can find and access by looking them up on the memory map.

Q. But why not do this is ROM space?

A. PET needs RAM type memory to store this data because it is always changing. The information in this "low" end of memory is actually produced by routines found in ROM.

Take for example the built-in clock. The clock or time is stored in locations 512, 513 and 514 (141, 142 and 143) in RAM. However, the data comes from a routine found in ROM. The time is of course always changing, therefore it must be stored in RAM. But because it is in RAM, you may also change it; either by setting TI or TI$ or you can POKE the above three locations. Try it.

Now, let's try some examples;

1. Location 226 (00E2 in HEX) holds the position of the cursor on the line. Try these;

```
POKE 226,20:?"PRINTS AT NEXT SPACE

?"123456789";:?PEEK(226)

(new ROMS)
POKE 198,20:?"PRINTS AT NEXT SPACE

?"123456789";:?PEEK(198)
```

2. Location 245 (216) stores the line the cursor is presently on (0 to 24). POKing this location will move the cursor to the specified line after a display execution. For example try;

```
?"A": POKE 245,10:?"B":?"C"

POKE 245,21-1:?"cu":POKE 226,20:?"PRINTS HERE"

(new ROMS)
?"A":POKE 216,10:?"B":?"C"

POKE 216,21-1:?"↑ ":POKE 198,20:?"PRINTS HERE"
```

The above will move the cursor to line 20 (21-1), print a 'cursor up' on line 21 and display your message starting at column 21, line 20.

While experimenting with out-of-range values I obtained some rather interesting results. Try POKing location 245 (216) with a number greater than 24, say 40 or 60, and hit the cursor up/down key a number of times. Also, experiment with unusual numbers in location 226 (198) such as ;

```
POKE 226,100:?"123456789"

(new ROMS)
POKE 198,100:?"123456789"
```

3. Location 526 (159) is the reverse field flag. POKing this address with a non-zero value will execute the following same line print statements in RVS field. Once finished, PET resets 526 (159) to zero.

Try this;

```
POKE 526, 1:?"123":?"456"

(new ROMS)
POKE 159, 1:?"123":?"456"
```

now INST a semi-colon between 3" and the colon (i.e. ...23";:?"4...) and re-execute.

4. Notice below the RVS field flag is location 525 (158), the number of characters in the keyboard buffer. Above the RVS flag is the buffer itself at locations 527 through 536 (623 - 632). Although this designates 10 buffer locations, there are actually only 9. The tenth 536 (632) is for some reason a "dead" location. During program execution, the operating system scans the keyboard every 60th of a second. If keys are typed say during a "FOR-NEXT" loop, they are stored in the keyboard buffer until the program encounters a GET or an INPUT*. PET then 'draws out' the contents of the buffer and implements them according to the command involved (GET or INPUT). However, if more than 9 keys are typed during the loop, PET erases the entire contents of the buffer and continues to fill the buffer with the 10th character as if it were the first, and so on ("modulo 10").

In the command mode (i.e. when you're operating PET directly) all typed keys go first into the keyboard buffer and then into screen memory or VIDEO RAM. However, you may also load the buffer under program control by POKing the ASCII representations of the characters into sequential locations of the buffer. You must also increment by 1 the contents of 525 (158) each time another character is POKed in, but remember - not past 9. Try the following endless loop. 145 is a cursor up;

```
POKE 525,4:POKE527,145:POKE528,145:POKE529,145:POKE530,13

(new ROMS)
POKE 158,4:POKE623,145:POKE624,145:POKE625,145:POKE626,13
```

Some other interesting items are;

old ROMS		new ROMS
POKE 59409,52	- blanks screen	"/9
POKE 59409,61	- screen back on	"/9
POKE 59468,14	- lower case mode	POKE 59468,14
POKE 59468,12	- graphics mode	POKE 59468,12
POKE 537,136	- disables STOP key and clock	POKE 144,49
POKE 537,133	- enable STOP key and clock	POKE 144,46

THE SYStem COMMAND

On the last three pages of memory map are listings of the subroutines in PET ROM that perform your commands and programs. These subroutines are stored as machine language. When a SYS command is executed PET jumps to the specified decimal address and continues from there in machine language.

* or after a BREAK, READY

Take for example the Machine Code Monitor program. This is a machine language program and is initialized by a SYS command stored as a BASIC program line. LOAD and RUN your M.L.M. then type 'X' and hit 'RETURN' to exit to BASIC. Now list. What you'll see is:

```
10  SYS (1039)
```

Location 1039 is the address to which PET will jump and also the address at which the first machine language instruction is stored. When this BASIC line is executed PET operates in machine code beginning with address 1039. New ROM PETs have the Machine Code Monitor located in ROM. It may be accessed by SYS 1024.

The SYS command does not require brackets around the specified address.

Since PET has its subroutines stored in machine language you can use the SYS command to access and execute them. However, you may come up with some rather peculiar if not disastrous results. When jumping into ROM you may find yourself in the middle of a subroutine or at the beginning of a subroutine belonging to a major function routine. Often PET will 'hang-up' or crash and you will be forced to power down to resume normal operation. To demonstrate jumping into the middle of a routine, try the following example;

```
SYS64824  (FD48)
```

The numbers on the right are the addresses of the above subroutines in hexadecimal. Compare them to the memory map, especially for e.g. # 1. Also take a look at 523.

The following are examples of valid locations which you can use with the SYS command to access useful routines, however, these routines are already accessible through BASIC.

1. SYS62651 (F346)

2. SYS63134 (F69E)

Example # 2 will perform a 'SAVE' but will not produce a tape header.

SUMMARY

This has been merely 'a scratch on the surface' of the extremely complex inner workings of PET. Do not be afraid to experiment with the POKE and SYS commands. There is absolutely nothing you can do to harm PET from the keyboard that turning power off and on won't fix. Also do some PEEKING around especially in low end memory. One good way is to write a small monitor program:

```
                10  ?" ♡ "PEEK(516):GOTO 10
(new ROMS)      10  ?" ♡ "PEEK(152):GOTO 10
```

The above will monitor the 'SHIFT' key. Try running it and depress 'SHIFT'. Now PEEK the low end of memory again.

When POKing or SYSing to random addresses, remember the address you choose. Often PET will do something which may erase the address from the screen (e.g. SYS64840). The addresses that have been listed here are only a few of many that are available for manipulation. Probe around and send in any discoveries, useful, peculiar or otherwise. They will be collected together and published in a future edition of CPUCN.

OWNERS REPORT - MIKE TODD

Mike's PET was one of the first 8K machines to arrive in the U.K., and the fault to which he refers (caused by frequent opening and closing of the PET casing) has not been reported for many months. In spite of early setbacks Mike became involved in his PET and developed some interesting techniques which are reproduced here. He has recently had his machine fitted with the new ROMS and his latest discoveries will be published in a future article.

I've had my PET for over 18 months, and in that time I have had more than my fair share of problems - but in that time I have also come to understand many of the PET's idiosyncracies. Could all of my problems have been due to the fact that it was delivered on March 13th 1978 at 1.13pm (1313 on the 24 hour clock) or that I was living in London W13 in a house number 52 (= 4 x 13) or even that its serial number is 1000013!!? I wonder....

Some of the most aggravating problems have been fairly simple, mainly due to dirty connectors. These faults included very severe screen judder, intermittent keyboard operation, intermittent cassette reading and writing, even the PET crashing completely - all were cured by careful cleaning of the internal connectors. I would certainly recommend periodic cleaning of these connectors (say once every six to eight months) with a commercial switch cleaner. The most important to keep clean is the main power connector (that's the one which comes from the mains transformer area), since this can put a high resistance path in the power supply, resulting in poor regulation and 'glitches' on the +5 volt rail. However, DON'T be tempted to clean the IC sockets - this would be asking for trouble! I would also add that, if your PET is transported a great deal, the connectors can work loose and may need the occassional push home.

Some models (like mine) have a minor fault which allows the cassette bracket to scrape on the low power lead to the logic board, eventually scraping away the insulation and shorting the supply to chassis. I can only suggest that this connector is bent very slightly backwards (very carefully) and the cassette bracket is pushed forward to keep it clear. A piece of insulating tape around the wires where they enter the plug might also help.

Surprisingly, I have little problem with the cassette unit, such as the reported partial erasure of tapes and azimuth errors, and can only attribute this to the careful cleaning of the heads now and again, as well as a total lack of tampering with the mechanics of the recorder.

The remainder of this article describes some of the software techniques that I have found useful, but I must point out that they are designed to work on the 8k machine, and therefore may not work on the 16/32k machines or on 8k machines with the new ROMs. If you are in any doubt, just try the routines out - you can't do any damage. If they don't work, then the routines and memory locations may have to be translated into the addresses

appropriate to the machine. I hope to do this myself very soon.

If you are bold enough to write machine language programs then you will appreciate the waste of time and space that DATA statements containing the machine code can take up. If your program resides in the second cassette buffer then it is possible to SAVE a composite version of your machine code and BASIC program. During development it is better to keep the two programs separate which will allow you to work on each program independantly of the other. However, when development is complete it is easy to SAVE a composite version.

First, LOAD the machine code program into the second cassette buffer, then clear memory by typing NEW and LOAD the BASIC program. All you need to do is to type the following:

```
A=PEEK(124) : B=PEEK(125) : C= number of characters in program name
POKE 249,0 : POKE 250,128 : POKE 229,A : POKE 230,B :
POKE 247,58: POKE 248,3 : POKE 238,C
```

Now, clear the screen and type the name of the program at the top of the screen keeping the shift key depressed. You'll get a load of gibberish if you are in graphics mode, but don't worry. Move the cursor down the screen a couple of lines and type SYS(63153) and the composite program will be SAVEd - from the start of the machine code program to the end of the BASIC program.

This program may then be LOADed and RUN in the normal way without any need for statements in the BASIC program to load the machine code program.

It might be worth adding that this technique will only work if the PET has already LOADed a program from cassette 1.

-oOo-

I have read a lot about the problems associated with leaving FOR..NEXT loops before the NEXT is encountered - this can cause problems because the FOR entry is left on the stack and can use up space until it is cleared by the final NEXT in the loop. However PET BASIC is rather more intelligent than it is given credit for. Any open FOR..NEXT loops appearing in a subroutine are closed, and their entries on the stack cleared, as soon as the RETURN is encountered. This is done because the GOSUB entry on the stack will occur before the FOR entries, and as soon as the RETURN is encountered the stack is cleared up to and including the GOSUB entry. In addition, whenever a new FOR.. NEXT loop is encountered, the stack is purged of any open FOR entries using the same variable - thus two FOR..NEXT loops using the same variable cannot be open at the same time. Thus, leaving open FOR..NEXT loops is very rarely

a problem provided that these points are borne in mind.

-oOo-

One of the greatest problems with the PET is the inability to communicate directly with your own output routines - the problem occuring when trying to pass the parameters to the routine. The most common method is to convert characters to numeric values using BASIC statements, and then pass these to an output routine using the USR(X) function. However, it is possible to pass parameters after the SYS(X) instruction - but more important, it is possible to use any expression (string or numeric) and pass the result to the output routine. Thus, if an output routine were sitting in the second cassette buffer, SYS(826) A$ would pass the string A$ to the routine while SYS(826) 4.3*A would pass the value of 4.3*A. The method is very straightforward and has many applications other than for output routines.

In the interpreter there is a routine (at CCB8) which will scan an expression, evaluate it and determine if the result is a string or a numeric value - if it is a string, the length and starting location of the string is stored; if it numeric, the result is placed in the floating point accumulator (FAC), and this can be converted to a string by using two more routines (DCAF & D36B). Finally, loose ends must be cleared up and the string pointers gathered together - this is done by the routine at D57E.

When all this is done, 0071 & 0072 hold the pointer to the start of the string and the accumulator holds the string length. Therefore it is easy to extract characters one at a time from the string and output them as appropriae. The following example illustrates the technique using the PET output routine at FFD2 which will output a character to the screen from the accumulator.

```
033A  20 B8 CC   JSR  CCB8       ; evaluate expression
033D  24 5E      BIT  005E       ; test for result type
033F  30 06      BMI  0347       ; branch if string variable
0341  20 AF DC   JSR  DCAF       ;  convert FAC to string
0344  20 6B D3   JSR  D36B       ;  get string pointers
0347  20 7E D5   JSR  D57E       ; clean up pointers etc.
034A  A0 00      LDY  #00        ; clear output pointer
034C  AA         TAX             ; X holds number of characters
034D  F0 09      BEQ  0358       ; branch if none
034F  B1 71      LDA  (0071)+Y   ; fetch character
0351  20 D2 FF   JSR  FFD2       ; and output it
0354  C8         INY             ; increment output pointer
0355  CA         DEX             ; count characters
0356  D0 F7      BNE  034F       ; branch if more
0358  60         RTS             ; return when finished
```

This routine acts in a similar way to PRINT, but does not allow formatting controls (e.g. TAB, comma etc.) or more than one expression.

The instruction JSR FFD2 could be replaced by any output routine, e.g. to drive a Selectric, or (as I use it) to drive a Prestel/Viewdata television which gives FULL COLOUR GRAPHICS and alphanumerics!! The majority of ASCII control characters can be passed using reverse field characters (e.g. reverse field 'A' return a value of 1 etc.); however, beware of using reverse field 'M' or 'O' since these can produce some odd results.

-oOo-

The use of reverse field characters allows the user to include DELete (reverse field 'T') and carriage return (reverse field shifted 'M') into strings. Whenever the character is encountered it is treated as a control character, even when LISTing - thus a line which ends with a string containing several reverse field 'T' will DELete itself on LISTing!! - this would allow various parameters to be 'hidden' from the general user.

-oOo-

Returning to machine code output routine; if your routine writes directly to the screen, you will be well aware of the snow effects that can be encountered. There are two ways of overcoming the effect; the first is to turn off the screen while writing (the PET does this every time it scrolls the screen) using the instructions LDA#34 STA E811, replacing the 34 with 3C will turn the screen back on again; the second is to force the program to wait until the video driver is not accessing the video RAM, (i.e. during vertical retrace) by waiting for the SYNC signal to go low. This can be done using the following routine which should immediately precede the output instruction:

```
WAIT  LDA  E840     ; get ORB
      AND  #20      ; isolate SYNC bit
      BNE  WAIT     ; loop until SYNC bit clear
```

This can be disabled by POKE 59458,44 which forces SYNC low so that it always appears to be in a retrace condition; POKE 59458,12 will restore operation to normal. Notice that the PET uses this waiting technique when outputing to screen and the disabling instruction can be used to speed up all PET output (program output and LISTing).

For those who output to screen in BASIC, WAIT 59456,32,32 has a similar effect in reducing snow at the cost of reduced output speed.

-oOo-

For those who are ambitious enough to make a very minor modification to the PET, I can recommend the inclusion of a small earpiece insert (of the type found in telephones - usually available through electronic junk shops) connected in series with a 20kohm resistor across the

cassette 1 'read' line. This provides useful information as to what the cassette is doing, and gives some idea of where you are on the tape. The addition of this has no adverse effect on the operation of the machine provided that the d.c. resistance is at least 20kohms. There is no reason why the earpiece should not be made switchable between the cassette line and one of the PET outputs so that PET music fiends can use it as a built-in loudspeaker. With practice it is possible to identify leader, header blocks, program blocks and data blocks as well as interblock spacer. It also allows verification that the cassette read/write is functioning.

-oOo-

For those who program in BASIC, it is often necessary to convert from decimal to hex (or some other base) and back again. The following one-liners will do this fairly efficiently:

DECIMAL to HEX (nb: D contains decimal number and H$ must be clear to "" before entry; H$ holds hex value on exit)

```
10  IF D THEN A=INT(D/16) : H$=MID$("0123456789ABCDEF", 1+D-A*16, 1)+H$:
        D=A: GOTO 10
```

HEX to DECIMAL (nb: H$ contains hex number, D exits with decimal value)

```
10 D=0 : IF H$>"" THEN FOR I = 1 TO LEN(H$): A=ASC(MID$(H$,I,1))-48:
                       D=D*16+A+(A>9)*7 : NEXT
```

These routines can be changed for any base merely by changing every occurance of '16' to the new base.

-oOo-

Many of you may have heard or read of the new improved 'PET user manual' which has been published. I managed to buy a copy some time ago and must say that I was very pleasantly surprised.

The new handbook (there appears to be a different one for each of the available machines) contains over 120 pages covering the basic operation of the PET, including detailled descriptions of what the interpreter is up to. For instance did you know that whenever FRE(0) is encountered, the PET does a massive garbage collection routine which clears up all string variables not being used, and makes as much space available as possible? Coverage is given to all aspects of BASIC programming together with a couple of example programs. There is also a section on use of the PET input/output ports, details of the IEEE bus and a complete listing of the machine language monitor which is available on cassette.

The appendix contains details of error messages, a detailed memory map (including page zero), details of variable storage and a reference list of BASIC statements,unfortunately many items are omitted from the list (such as string operators) which makes the reference incomplete. There is also plenty of space for your own

notes. With the final appendices covering how to conserve time and space, a parts list and some suggested reading (USA oriented), I can recommend this handbook as essential to anyone who takes PET programming seriously. I might also add that this is an unsolicited testimonial!!

-oOo-

Finally, going back to the plethera of '13's in the life of my PET, I was shocked to discover that my PET was 13 months old on Friday 13th April 1979!! I wasn't superstitious, but now, well.....

8K - 16/32K COMPARISON

Collected by Jim Russo

The Operating System of the 16/32K PET is about 90% the same as the 8K PET, but has been re-assembled so that almost everything is in a slightly different place in memory than it used to be. Most bugs have been fixed and some limitations removed.

Any pure BASIC program (no PEEK, POKE, SYS, or WAIT) that works on an 8K PET should also work on a 16/32K PET. POKing and PEEKing screen memory (32768 to 33767) is still safe but POKing the operating system (below 1024 decimal) or using an operating system PEEK value to make a decision could be hazardous. Other programs can be made to work properly if references to RAM and ROM locations are changed. Commodore's 16/32K PET manual contains a memory map for pages 0. 1 and 2. A list of new ROM addresses follows. These two lists should contain the information needed in most cases.

Main Hardware Differences:

- The character generator ROM has been revised so that when lower case mode is selected, upper and lower case are interchanged. That is, the 'SHIFT' key must be used to obtain an upper case character. Also, 8K programs using lower case that are run on a 16/32K PET will display all lower case as upper case and vice versa.

- The signal which blanks the video on the 8K is not connected on the 16/32K, so POKE 59409, 52 no longer works. The ROM routines still reference this address but the required hardware seems to have been omitted.

Summary of Differences

- The bug in TI has been fixed. Now every 623rd. interrupt doesn't increment TI. Also, TI is allowed to count 1/60 sec. too far: 240000 precedes 000000.

- Execution (of at least some code) is faster due to more efficient coding and better use of zero page. PRINT (to screen) is faster because extra code to maintain separate POS pointer has been eliminated. Also, screen snow and 'scroll - up flash' has been eliminated thanks to dynamic screen RAMs.

- Standard typewriter operation i.e. shift for upper case.

- RND (0) returns a number derived from interval timers.

- OPENing more than 10 files no longer crashes system.

- OPEN statement correctly sets "current tape buffer pointer".

- Machine Language Monitor included in ROM. BRK vector is initialized to monitor. 'L' and 'S' (LOAD and SAVE from monitor) have new syntax.

* - NMI vector no longer tied to +5v. NMI is initialized to BASIC "Warm Start".

- Data file write error corrected. The Tape Output routines now wait 2/3 second after turning on motor before beginning to write tape leader. 8K PET waited 13ms. on drive 1, 57 ms. on 2.

- Cursor home, left, right, up, down are now tracked properly by the POS function. This causes apparent differences in the TAB function which subtracts POS from its argument to determine the number of spaces needed.

- SPC (0) corrected.

- When output is directed to an alternative device, the ASCII space code $20 is used for all BASIC supplied forward spacing. 8K used $1D.

* - Screen blanking (POKE 59409,52) no longer available, however, the scroll routine still uses it as if it did. Note some control line is still used for "EoI OUT".

- PEEK is no longer restricted.

- Array dimensions now as high as 32767 (used to be 256).

* - The memory expansion port uses a different connector.

- Spaces no longer allowed in keywords (e.g. GOSUB cannot be coded as GO SUB).

- POKE and PEEK can now be used in the same line (i.e.,POKE 8000, PEEK (9000) now works).

- ST (the status word) if used, must be tested before input of file data.

- Most ROM routines and RAM addresses have changed.

	8K	16/32K
INTFLP	D278	D26D
FLPINT	D0A7	D09A
CHRGOT	00C2	0070
WARM START	C38B	C389
FLOATING AC	00B0-00B6	005E-0064

- BASIC input buffer is no longer in zero page so programs which used many free locations in this area must be re-written.

* - The decoding of screen memory now uses A11 (address buss line 11). Addresses 8800-8FFF (34816 to 36863 decimal) no longer address screen memory.

(* HARDWARE CHANGES)

BAN THE BOMBOUT

E.A. Anderson,
Channel Data Systems,
5960 Manderin Avenue,
Goleta CA 93017,
U.S.A.

Much has been written regarding the use of GET rather than INPUT to avoid dropping out of a program and still retain the editing features of INPUT. There is a very simple way to use INPUT and avoid dropping out of the program. Just open a file to the screen (OPEN #,3) at the beginning of your program; then add a CMD # before each input statement. Select the file number (#) to be unique from other files you may open later.

There are some other useful features of this approach. If a prompt string is printed on the same line as the entry the program will not advance when you hit RETURN unless an entry is made. If the prompt is made on the line before the entry the input will be zero for numeric entries and a null for string entries. The prompt question mark will not be present unless you include it in the prompt string. Multiple entries can be entered individually with RETURNS after each entry or with a comma separating each entry. After each RETURN the cursor will advance a space on the same line.

To exit the program do a LOAD/RUN (shifted RUN/STOP). It is best to use only string entries in any input statement to avoid dropping out of the program if the operator enters a string in response to numeric entry request. If LOAD/RUN is pressed when only string entries are used you will get a nice BREAK IN LINE #, almost as if it were designed to be used that way.

We have incorporated this feature in our latest versions of Personal Ledger, Data Logger and our new Omnifile data base software. We hope others will also use it to help Ban the Bombout.

Sample code and output

```
10  OPEN7,3
20  REM
30  REM YOUR PROGRAM
40  REM
50  CMD7,;:INPUT"NUMBER,NAME? ";N,N$:PRINT#7
60  PRINT N,N$
70  GOTO50
```

The ';' after 'cmd7,' and the 'PRINT#7' in line 50 are optional. They suppress the line feed after 'CMD7,' and add a line feed after the entry, respectively. Sample output is given below. An 'r' indicates a carriage return has been pressed. The boldface type indicates operator entries.

```
RUN

NUMBER, NAME? 1,JOHNr
  1         JOHN
NUMBER, NAME? 2,r
  2
NUMBER, NAME? JANE,r
  0         JANE
```

```
NUMBER, NAME? ,r
  0
NUMBER, NAME? 6rGEORGEr
  6          GEORGE
NUMBER, NAME? rrrrrrrrrrrrrrr4,JILLr
  4          JILL
```

An alternative form of line 50 which gives identical results is:

```
50 CMD7,"NUMBER, NAME? ";:INPUTN,N$:PRINT#&
```

However, if either the prompt string or the ';' after the prompt string is removed, a zero will be entered for the number of a null for the string when RETURN is pressed after each entry as illustrated below:

```
50 CMD7,;:INPUTN,N$:PRINT#7
gives:

NUMBER, NAME? rr
  0

50 CMD7,"NUMBER, NAME? ":INPUTN,N$:PRINT#7
gives:

NUMBER, NAME?
rr
  0
```

HARDWARE HINTS

Video Off

Bill Williamson

Psychologists working in the field of Perception have made good use of the 8K PET's facility to switch the video screen "on" and "off". Now Bill Williamson, Chief Electronics Technician to the Department of Psychology at Leicester University has come up with a hardware modification which provides this facility on the new 16K and 32K machines.

Note - Hardware modification to PET will void your warranty. This particular modification may cause unexpected side-effects if your PET is used with other IEEE peripherals.

This modification allows the use of POKE 59409, 52 (BLANK) and POKE 59409, 60 (UNBLANK) on the new 16K and 32K PETs.

a) Locate I.C. 74LS20 which is at grid reference G11 on main P.C.B.

b) Cut pin 1 from the +5V rail and link it to PIN 39 on PIA #1 which is the BLANK TV OUTPUT.

Note. PIA # IS THE 6520 NEAREST THE MAINS TRANSFORMER.

QUIETER WRITER

There are some wing nuts in the bottom of the Tractor Printer. These are only there to secure the Printer mechanism in transport and therefore are not required for general use of the printer.

Their removal will make the operation of the printer quieter. We recommend you keep the wing nuts in a safe place in case you wish to transport your printer around at any time.

PET BITS

Latest products to become Commodore Approved which should be available at your local dealer, include a Bar Code Reader from Machsize (details available from Duncan Smythe on 0926 312542/32399) and Trader Pac, a fully integrated Invoicing, Stock Control and Sales Ledger from Bristol Software Factory (details available from John Kyle-Price on 0272 20801)

* * * * * * * * * * * *

Nick Vine announces a machine code program to drive the PR40 as a page printer at full speed. The program costs £12 and is available from him at 61, Mortlake Road, Key, Surrey. TW9 4AA

* * * * * * * * * * * *

Logic Box have released a mains interference suppressor to protect your PET and floppy disk unit from sudden mains spike's (which can corrupt your data and damage your hardware). The units are priced at £19.oo (inc. VAT and P&P) and are available from their offices at 31, Palmer Street (off Caxton St.) LONDON SW1. Quantity discounts are available.

* * * * * * * * * * * *

"Cursor" a games oriented magazine that comes as a series of programs recorded on cassette is now being distributed in this country by PETSOFT (further details on 0635 201131)

Two booklets introducing PET techniques for raw beginners "The PET Personal Computer for Beginners" have been written by Seamus Dunn and Valerie Morgan. (further details from them at The New University of Ulster, Coleraine, Co. Londonderry, Northern Ireland. BT52 1SA)

* * * * * * * * * * * *

Watching a cassette load

Jim Butterfield
Toronto

It may not be too useful, but it's very satisfying to watch a program coming in from cassette tape. Much of what comes in will look like gibberish, since the program contains obscure things like pointers, flags and tokens. But it's interesting to see, and here's how you can do it.

Step 1: load any Basic program on cassette 1. The program doesn't matter; the LOAD activity sets up certain internal things that will help us.

Step 2. set up the cassette with any BASIC program ready to load. A short one would be good: that way you may catch the whole program on the screen. But any BASIC program will do.

Step 3. set graphic mode with POKE 59468,14. This may help you spot a few recognizable pieces of your program.

Step 4. Give SYS 62894. PET will ask you to press PLAY. Do so, and in twenty seconds or so, PET will report FOUND ..... and stop.

Step 5. Clear the screen so you'll get a better view of the program as it comes in. Now move the cursor down to a few lines from the bottom of the screen.

Step 6. enter POKE 636,128:POKE 638,132:SYS 62403.

Step 7. sit back and watch the program load to the screen. You won't be able to run it, of course, since it's in the wrong part of memory... but isn't it fascinating to watch?